Handling Breakdowns in Unmanned Aircraft Systems

Patrice Carle, Christine Choppy, Romain Kervarc, Ariane Piel

Handling Breakdowns in Unmanned Aircraft Systems

Patrice Carle[1], Christine Choppy[2], Romain Kervarc[1], and Ariane Piel[1]

[1] ONERA — The French Aerospace Lab, 91123 Palaiseau, France
[2] LIPN, UMR CNRS 7030, Université Paris 13, 93430 Villetaneuse, France

Abstract. This work is devoted to activity recognition in the setting of data analysis in aeronautics. Formal methods are applied to the certification and safety analysis processes of Unmanned Aircraft Systems in breakdown situations. The behaviour of these systems in case of a failure is entirely modeled and implemented. A temporal language — the Chronicle language — describes arrangements of events which are employed to detail undesired circumstances that would lead to breaches in safety. A C++ chronicle recognition tool is used to recognise all the possible occurrences of these situations as soon as they occur.

Keywords: safety monitoring, behaviour recognition tool, industrial applications, unmanned aircraft systems

1 Introduction

One obstacle to the insertion in controlled or uncontrolled airspace of aircrafts without pilots onboard results from security issues linked to the global consistency of the system. In the framework of operation safety analysis, we assess the possibility to detect incoherent states between the different entities making up the system. We formalise these different incoherent states in order to be able to automatically recognise them, and hence offer the opportunity of a self-acting surveillance. More specifically, this ongoing work relies on a fragment of the IDEAS project in charge of the Insertion of Unmanned Aircrafts in Airspace and Security, and tackles consistency problems in breakdown handling policies for Unmanned Aircrafts (UA).

An UA is defined by the FAA (Federal Aviation Administration) as "an aircraft that is operated without the possibility of direct human intervention from within or on the aircraft". The system required to operate safely an UA is referred to as an Unmanned Aircraft System (UAS). There exists several types of such architectures, and our model is based on the one presented in Fig. 1. It is composed of three entities, the UA, the Remote Pilot Station (RPS), and the Air Traffic Control (ATC). All three interact via several communication links. The RPS pilots the Unmanned Aircraft via Telecommand (TC), and the Unmanned Aircraft sends information to the pilot through Telemetry (TM). In addition, the ATC and the pilot can communicate via radio (Voice) relayed by the UA.

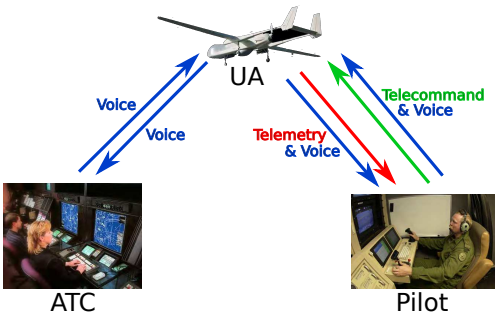

Fig. 1. Architecture of an Unmanned Aircraft System.

Hence, the dynamic data flows between the agents of the system and between different systems if several UAS are considered are very elaborate. Moreover, each agent deduces from its own observations the state of the other agents. The situation in case of a fault can therefore be very complex. Formal methods are well-suited for monitoring UAS because these systems are highly automated as well as critical so require strong risk-free guarantees.

In the framework of the IDEAS project, the behaviour of each entity in case of a failure has been specified [9]. Our work focuses on overseeing the consistency between the three entities during these rundown story lines.

Consider for instance the following situation: an unmanned aircraft which telecommand has been poorly working and then completely lost. The conforming procedure has been followed correctly, and the ATC has activated the urgency mode corresponding to the rerouting of the aircraft. However, the telecommand is suddenly restored. The pilot, busy redirecting the UA back to its original track, forgets to warn the ATC to cancel the urgency procedure. Thus, the ATC considers the aircraft to be rerouting itself to a given diversion airport while it is actually taking another route. It is important to be able to identify this situation which is critical since the ATC, lacking the appropriate information, might not properly organise the flow of traffic to separate aircrafts and prevent collisions.

2 A Preliminary Semi-Formal Representation

The aim of this work is to provide a tool for automatic monitoring of consistency in UAS. It is therefore necessary to formally set up the problem. The situation is standardised into the UML norm [10] as follows.

As a first step, a class diagram is built describing precisely the structure of the UA as shown in Fig. 2. Each UA X (`UA(X)`) is seen as made up of several components: the telecommand (`UA(X) TC`), the telemetry (`UA(X) TM`), the autopilot (`UA(X) Pilot`), the radio relay (`UA(X) Relay`), and the transponder code it sends to the ATC (`UA(X) Code`). Similarly, the associated RPS is composed of the telecommand (`RPS(X) TC`), the telemetry (`RPS(X) TM`) and a connection to

the ATC (`RPS(X) − ATC connection`). Note that the smooth functioning of each end of a given communication link is considered separately so as to be able to differentiate the state of each entity to identify the source of potential problems and check for the consistency between the states. For example for the telecommand, the emission of instructions from the pilot, `RPS(X) TC`, and the reception of these instructions by the UA, `UA(X) TC`, are distinguished.

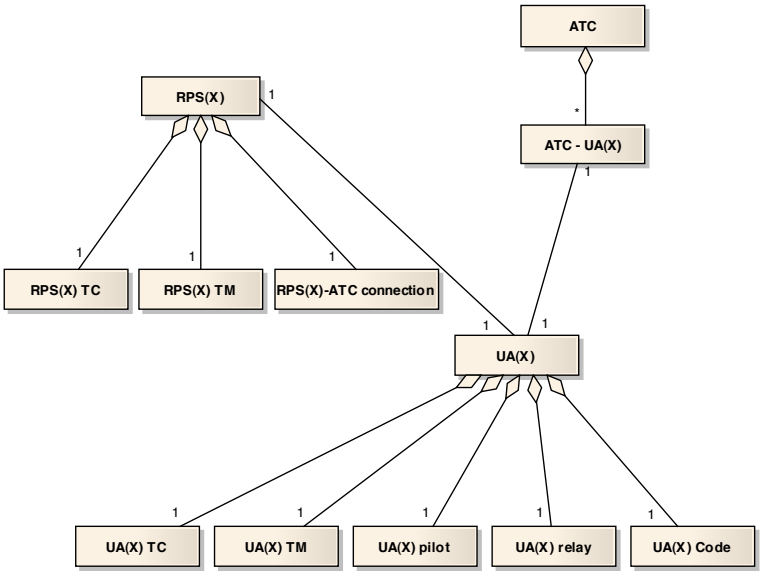

Fig. 2. Class diagram describing the structure of an UA.

Secondly, so as to exploit the codified behaviours specified in the IDEAS project, these were translated into state-transition diagrams for each fault. The first breakdown studied is TC failure presented in Fig. 3: the pilot receives information from the UA via telemetry but cannot send out orders to it.

With such diagrams, the system is entirely modeled. Its life cycle is mirrored by the changes in the active states of the diagram. In order to allow direct simulation, the UML diagram is implemented in C++ using the Meta State Machine (MSM) library [8] of boost (Version 1.48.0) which provides a straightforward way to define state machines. So as to simulate the life cycle of the system, scenarios activate the transitions of the diagram.

3 A Recognition Tool

In Sec. 2, the simulation of the system that has to be overseen is described. This section will now deal with the issue of how to put in place this supervision

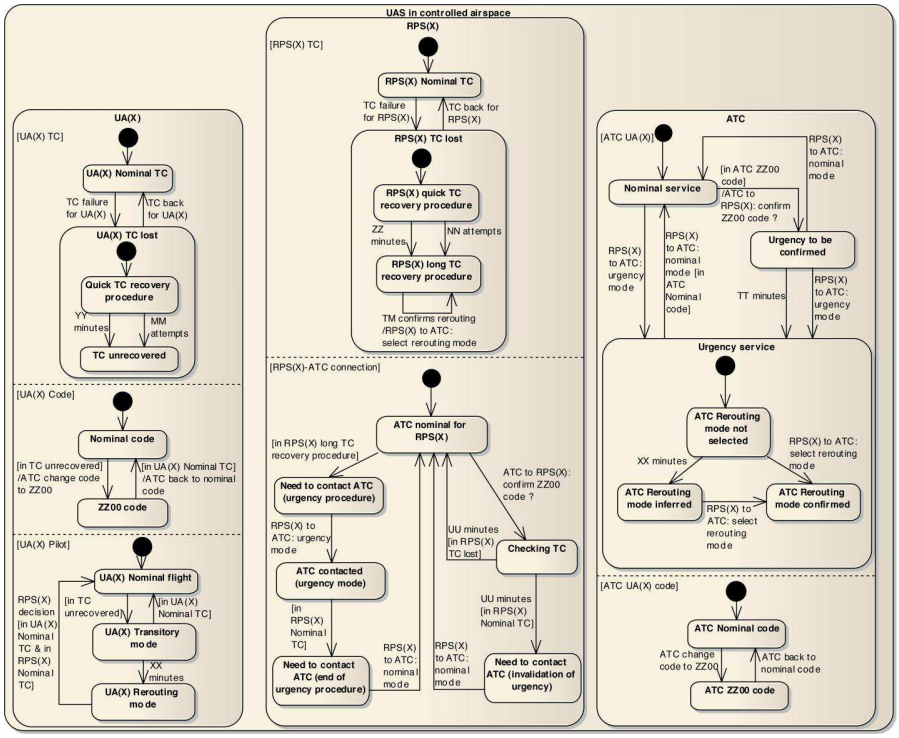

Fig. 3. State-transition diagram describing the system behaviour in case of TC failure.

of the system, the aim being to detect during the simulation critical situations previously determined by experts.

A temporal language — the chronicle language [5] — is used to formally describe these system behaviours to oversee the consistency between the three entities of the UAS. It formally describes arrangements of events: a chronicle is either a single event, the conjunction of two chronicles, the disjunction of two chronicles, the sequence of two chronicles or the absence of a chronicle during another chronicle. For instance, let `E`, `F` and `G` be single events, chronicle (`E F`) − [`G`] corresponds to the sequence `E F` (event `E` followed by event `F`) without event `G` occurring in between `E` and `F`.

This formal language enables us to precisely detail interesting (in our case undesired) arrangements of events by writing chronicles. Different examples of applications of this concept may be seen in [6], and [7] offers another approach dealing with a temporal logic for unmanned aircrafts, but in an execution, rather than safety, monitoring framework. Subsequently, once appropriate chronicles are established, all that is left to do in order to supervise the consistency of the system is to watch for potential recognitions of these chronicles in the event flow.

Table 1. Chronicles overseeing the consistency of the system in case of TC failure.

RPS rushed decision (((to_UA(X)_nominal_flight || to_UA(X)_transitory_mode) TT) −[to_UA(X)_rerouting_mode]) & to_ATC_rerouting_mode_confirmed *where TT is a given delay.* *The* ATC *is in* rerouting mode confirmed *but* UA(X) *is not in* rerouting mode.
ATC late (to_TC_unrecovered TT) −[to_Urgency_service || to_UA(X)_nominal_TC] *where TT is a given delay.* UA(X) TC *changed to* TC unrecovered *and since then* TT *minutes have elapsed without the* Urgency service *of the* ATC *being an active state.*
ATC incoherent to_UA(X)_rerouting_mode ((to_ATC_rerouting_mode_inferred D) −[to_ATC_rerouting_mode_confirmed || to_UA(X)_nominal_flight]) *where* D *is a given delay.* UA(X) *changed to rerouting mode. Following that, the* ATC *inferred the rerouting mode but it was not confirmed by the pilot even after a delay of* D *minutes.*

In our work, the set of events considered to build these chronicles are the entrances in the different possibly active states of the diagram of Fig. 3. Table 1 provides a few examples of chronicles overseeing the consistency of the system in the case of TC failure. The first chronicle "*RPS rushed decision*" corresponds to the example presented in Sect. 1. Chronicle "*ATC late*" could be triggered by the following circumstances. Consider an UA which has been periodically losing its TC link, for instance because of a disruptive environment, and the urgency transponder code associated to TC failure has been coming on and off. Both the pilot and the ATC are tired out and less alert, so when the TC is suddenly lost for good, no one reacts. After a significant delay predefined in its flight plan, the UA starts to reroute to a diversion airport. If neither the pilot nor the ATC realise the change in situation other aircrafts might not be sufficiently separated from the UA which cannot be sent orders to anymore. Recognising the chronicle would have prevented the possibly dangerous outcome.

For a given chronicle and a given stream of events, we are interested in establishing the list of all its recognitions gradually as events flow. This requirement prevents using simple finite state automata [3]. With this design, a recognition tool called Chronicle Recognition System (CRS/ONERA) has been developed by the ONERA in the late 1990s [4]. It is designed on the basis of duplicating automata so as to comply with performance and interoperability requirements.

In the framework of this Ph.D. thesis, it is interesting to develop a new recognition tool that directly results from the set semantics of the chronicle language, since the recognitions produced by this tool are therefore adequate by construction and no adequacy proof is necessary. This tool, called Chronicle Recognition Library (CRL), is implemented in C++. Chronicles are plugged into the program, and then, gradually as events flow in, the program gives the set of all the recognitions of each chronicle, specifying for each recognition which events lead to it.

In this implementation, the events are represented as triplets (`name`, `date`, `order`) where `date` corresponds to the date of occurrence, and `order` is distinct for each event, allowing to sort and order the buffer of events to be processed. Indeed, two events can occur at the same `date` but their `order` is necessarily different. For instance, two sensors can observe the same system and hence generate events at the same instants but these will be received by the observing system in a given order.

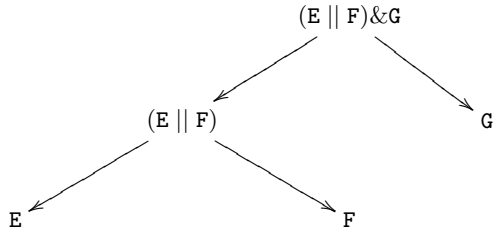

Fig. 4. Syntactic tree corresponding to chronicle (E || F)&G labelled with sub-chronicles.

A recognition of a given chronicle is a set of events leading to it. The recognition method will be illustrated by chronicle (E || F)&G on the buffer of events $\{(e, d_e, 1), (h, d_h, 2), (g, d_g, 3), (f, d_f, 4)\}$. The program processes events from the event buffer in their order. The recognition process is defined by induction for each class of chronicle. A key notion in this recognition system is that of sub-chronicles shown in Fig. 4.

The chronicle to be recognised is seen as a binary tree, the nodes and leaves of which are respectively operators and single events. The process computes two sets for every node — and hence for every sub-chronicle — after each event: the overall recognition set of the sub-chronicle, and its new recognitions set. In Table 2, both these sets — the new recognitions in green — are presented for each sub-chronicle and after each event of the buffer. The new recognitions set is emptied after each processing of an event. It is used to make sure to only add new recognitions to the recognition set — and hence avoid doubles — and to determine if there will be new recognitions in the ancestors of the node.

The recognition set of the chronicle is the recognition set of the root of the tree. In our example, after having processed all four events, there are two recognitions and the final recognition set is $\{\{(e, d_e, 1), (g, d_g, 3)\}, \{(g, d_g, 3), (f, d_f, 4)\}\}$. Note that event $(g, d_g, 3)$ is in both recognitions.

Table 2. Evolution of the recognition sets for chronicle (E || F)&G.

	$(e, d_e, 1)$	$(h, d_h, 2)$	$(g, d_g, 3)$	$(f, d_f, 4)$
E	$\{\{(e, d_e, 1)\}\}$ $\{\{(e, d_e, 1)\}\}$	$\{\{(e, d_e, 1)\}\}$ $\{\}$	$\{\{(e, d_e, 1)\}\}$ $\{\}$	$\{\{(e, d_e, 1)\}\}$ $\{\}$
F	$\{\}$ $\{\}$	$\{\}$ $\{\}$	$\{\}$ $\{\}$	$\{\{(f, d_f, 4)\}\}$ $\{\{(f, d_f, 4)\}\}$
E || F	$\{\{(e, d_e, 1)\}\}$ $\{\{(e, d_e, 1)\}\}$	$\{\{(e, d_e, 1)\}\}$ $\{\}$	$\{\{(e, d_e, 1)\}\}$ $\{\}$	$\{\{(e, d_e, 1)\}, \{(f, d_f, 4)\}\}$ $\{\{(f, d_f, 4)\}\}$
G	$\{\}$ $\{\}$	$\{\}$ $\{\}$	$\{\{(g, d_g, 3)\}\}$ $\{\{(g, d_g, 3)\}\}$	$\{\{(g, d_g, 3)\}\}$ $\{\}$
(E || F) &G	$\{\}$ $\{\}$	$\{\}$ $\{\}$	$\{\{(e, d_e, 1), (g, d_g, 3)\}\}$ $\{\{(e, d_e, 1), (g, d_g, 3)\}\}$	$\{\{(e, d_e, 1), (g, d_g, 3)\}, \{(g, d_g, 3), (f, d_f, 4)\}\}$ $\{\{(g, d_g, 3), (f, d_f, 4)\}\}$

The current work on the development of CRL is to take into account delay constraints, *e.g.* `TT` in chronicle *ATC late*. Mirroring Allen's interval logic [1], delays are added to the chronicle language to be able to express duration of chronicles (`lasts` δ, `at most` δ, and `at least` δ — where δ is a delay) and delays after chronicles (`then` δ). It is clear that the system must be observed at each occurrence of an event. However, for chronicle C `then` δ, this is not enough and the question of when to examine the system is raised. Indeed, it cannot be constantly looked over. This is a general issue once continuous time and discrete events have to coexist. Different approaches have been put forward to address this problem, notably [2].

A first necessary step for this ongoing work is therefore to define a function which, for both a given chronicle and the past events that have been observed, returns the next time at which the system has to be checked for a new recognition. This can be seen as a sort of validity contract, which can be directly linked to distributed simulations since these carry messages with validity domains. This question of validity is indeed strongly present in distributed simulations for aeronautics but also in certain formal methods for dynamic systems.

4 Conclusion and perspectives

In this paper, a temporal language is used to monitor safety for Unmanned Aircraft Systems in case of breakdowns, which is a problem of particularly great importance in the field of civil aviation. The behaviour of the UAS is completely modeled in a UML diagram which is implemented in C++, and a C++ library allows direct analysis of simulation data.

Chronicles and the associated recognition model shall be extended to improve the expressivity of the framework, in particular with delay-related constructs inspired from interval logics. The UAS application previously exposed tackles the crucial problem of the insertion of UAS in controlled or uncontrolled airspace and

the associated certification and safety analysis processes. As presented in this paper, chronicle recognition can be used as a crucial tool to analyse simulation as the design stage of an UAS. It is also well-suited for the on-line analysis of real-time situations to generate alarms and avoid risky situations. This first step deals with physical failures in one aircraft but there are other issues, as separation and collision avoidance which are the cornerstone of flight safety. A model for this is currently being developed in collaboration with UAS engineers. Chronicles, already applied to a wide variety of different fields, are shown to be an adequate generic means to represent knowledge in a multi-agent system, and, as such, benefit to a large spectrum of applications. Moreover, the insertion of UAS into general airspace raises concerns that cannot be solved without a formal representation which chronicles seem to fulfil fairly.

Acknowledgements.

The authors would like to thank J. Bourrely for his help and support for this application, and T. Lang and C. Le Tallec for their useful insights on Unmanned Aircrafts.

References

1. Allen, J.F.: Maintaining knowledge about temporal intervals. Commun. ACM pp. 832–843 (1983)
2. Bennett, B., Galton, A.P.: A unifying semantics for time and events. Artificial Intelligence 153, 13 – 48 (2004)
3. Bertrand, O., Carle, P., Choppy, C.: Chronicle modelling using automata and colored Petri nets. In: The 18th International Workshop on Principles of Diagnosis (DX-07). pp. 229–234 (2007)
4. Carle, P., Benhamou, P., Dolbeau, F.X., Ornato, M.: La reconnaissance d'intentions comme dynamique des organisations. In: 6èmes Journées Francophones pour l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents (JFIADSMA'98) (1998)
5. Carle, P., Choppy, C., Kervarc, R.: Behaviour recognition using chronicles. In: Proc. 5th IEEE International Symposium on Theoretical Aspects of Software Engineering. pp. 100–107 (2011)
6. Carle, P., Choppy, C., Kervarc, R.: Detecting behaviours within HLA distributed simulations with added analysis components. In: 33rd IEEE Aerospace Conference (2012)
7. Doherty, P., Kvarnström, J., Heintz, F.: A temporal logic-based planning and execution monitoring framework for unmanned aircraft systems. Autonomous Agents and Multi-Agent Systems pp. 332–377 (2009)
8. Henry, C.: "MSM library of boost". `www.boost.org/doc/libs/1_48_0/libs/msm/doc/HTML/index.html` (2011)
9. Lang, T.: IDEAS–T1.1: Architecture de système de drone et scénarios de missions. Tech. rep. (2009)
10. "OMG Unified Modeling Language^{TM}(OMG UML), Superstructure, Version 2.4.1" (2011)